Derek Castro

04/13/2025

CS300

Module 6-2: Project One

**Vector Pseudocode**

FUNCTION main()

**// Load data from file**

CREATE an empty list to store courses

OPEN the file

IF file cannot be opened

PRINT "Error: Unable to open the file."

EXIT PROGRAM

WHILE there are more lines in file

READ next line

IF line is empty or invalid

PRINT "Error: Invalid line in file."

CONTINUE to next line

SPLIT line by commas into tokens

IF number of tokens is less than 2

PRINT "Error: Invalid format. Each line must have a course number and a course name."

CONTINUE to next line

CREATE a new course object

SET course number to first token

SET course name to second token

IF more than 2 tokens

FOR each token after the second one

IF token is empty or invalid

PRINT "Error: Invalid prerequisite."

CONTINUE to next token

APPEND token to course’s prerequisites list

APPEND course to list

CLOSE file

**// Validate prerequisites**

CREATE a list of all course numbers

FOR each course in list

APPEND course number to list of course numbers

FOR each course in list

FOR each prerequisite in course’s prerequisites

IF prerequisite is not in list of course numbers

PRINT "Error: Invalid prerequisite. Does not exist."

EXIT PROGRAM

**// Menu System**

WHILE true

PRINT "Menu:"

PRINT "1. Load Data"

PRINT "2. Print Course List”

PRINT "3. Print Course Details"

PRINT "9. Exit"

GET user choice

IF choice == 1

// (Existing load logic above)

ELSE IF choice == 2

CALL printSortedCourses()

ELSE IF choice == 3

PROMPT "Enter course number:"

GET searchTerm

CALL printCourseDetails(searchTerm)

ELSE IF choice == 9

EXIT PROGRAM

ELSE

PRINT "Invalid choice."

**// Print Sorted Courses**

FUNCTION printSortedCourses()

CREATE temporary list = original list

SORT temporary list by course number

FOR each course in temporary list

PRINT course.courseNumber + ": " + course.title

**// Print Course Details**

FUNCTION printCourseDetails(searchTerm)

FOR each course in list

IF course.courseNumber == searchTerm

PRINT course.title

IF course.prerequisites is not empty

PRINT course.prerequisites

RETURN

PRINT "Course not found."

**Hash Table Pseudocode**

FUNCTION main()

**// Load data from file**

CREATE empty hash table to store courses

OPEN the file

IF file cannot be opened

PRINT "Error: Unable to open the file."

EXIT PROGRAM

WHILE there are more lines in file

READ next line

IF line is empty or invalid

PRINT "Error: Invalid line in file."

CONTINUE to next line

SPLIT line by commas into tokens

IF number of tokens is less than 2

PRINT "Error: Invalid format. Each line must have a course number and a course name."

CONTINUE to next line

CREATE a new course object

SET course number to first token

SET course name to second token

IF more than 2 tokens

FOR each token after the second one

IF token is empty or invalid

PRINT "Error: Invalid prerequisite."

CONTINUE to next token

APPEND token to course's prerequisites list

INSERT course into hash table using course number as key

CLOSE file

**// Validate prerequisites**

FOR each course in hash table

FOR each prerequisite in course's prerequisites

IF prerequisite is not a key in hash table

PRINT "Error: Invalid prerequisite. Does not exist."

EXIT PROGRAM

**// Menu System**

WHILE true

PRINT "Menu:"

PRINT "1. Load Data"

PRINT "2. Print Course List"

PRINT "3. Print Course Details"

PRINT "9. Exit"

GET user choice

IF choice == 1

// (Existing load logic)

ELSE IF choice == 2

CALL printSortedCourses()

ELSE IF choice == 3

PROMPT "Enter course number:"

GET searchTerm

CALL printCourseDetails(searchTerm)

ELSE IF choice == 9

EXIT PROGRAM

ELSE

PRINT "Invalid choice."

**// Print Sorted Courses**

FUNCTION printSortedCourses()

CREATE temp list

FOR each bucket in hash table

FOR each course in bucket

ADD course to temp list

SORT temp list by course number

FOR each course in temp list

PRINT course.courseNumber + ": " + course.title

**// Print Course Details**

FUNCTION printCourseDetails(searchTerm)

SEARCH hash table for course with key == searchTerm

IF course is found

PRINT course.title

IF course.prerequisites is not empty

PRINT course.prerequisites

ELSE

PRINT "Course not found."

**Binary Search Tree Pseudocode**

FUNCTION main()

**// Load data from file**

CREATE empty binary search tree to store courses

OPEN the file

IF file cannot be opened

PRINT "Error: Unable to open the file."

EXIT PROGRAM

WHILE there are more lines in file

READ next line

IF line is empty or invalid

PRINT "Error: Invalid line in file."

CONTINUE to next line

SPLIT line by commas into tokens

IF number of tokens is less than 2

PRINT "Error: Invalid format. Each line must have a course number and a course name."

CONTINUE to next line

CREATE a new course object

SET course number to first token

SET course name to second token

IF more than 2 tokens

FOR each token after the second one

IF token is empty or invalid

PRINT "Error: Invalid prerequisite."

CONTINUE to next token

APPEND token to course's prerequisites list

INSERT course into binary search tree using course number as key

CLOSE file

**// Validate prerequisites**

FOR each course in binary search tree

FOR each prerequisite in course's prerequisites

IF prerequisite is not found in binary search tree

PRINT "Error: Invalid prerequisite. Does not exist."

EXIT PROGRAM

**// Menu System**

WHILE true

PRINT "Menu:"

PRINT "1. Load Data"

PRINT "2. Print Course List"

PRINT "3. Print Course Details"

PRINT "9. Exit"

GET user choice

IF choice == 1

// (Existing load logic)

ELSE IF choice == 2

CALL printSortedCourses()

ELSE IF choice == 3

PROMPT "Enter course number:"

GET searchTerm

CALL printCourseDetails(searchTerm)

ELSE IF choice == 9

EXIT PROGRAM

ELSE

PRINT "Invalid choice."

**// In-order traversal**

FUNCTION printInOrder(node)

IF node is null

RETURN

CALL printInOrder(node.left)

PRINT node.courseNumber + ": " + node.title

CALL printInOrder(node.right)

**// Print Sorted Courses**

FUNCTION printSortedCourses()

IF root is null

RETURN

CALL printInOrder(root)

**// Print Course Details**

FUNCTION printCourseDetails(searchTerm)

current = root

WHILE current is not null

IF current.courseNumber == searchTerm

PRINT current.title

IF current.prerequisites is not empty

PRINT current.prerequisites

RETURN

ELSE IF searchTerm < current.courseNumber

current = current.left

ELSE

current = current.right

PRINT "Course not found."

**Pseudocode Runtime and Analysis**

**Vector Runtime Analysis**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| FOR each course in list | 1 | n | n |
| IF course == searchTerm | 1 | n | n |
| PRINT course data | 1 | 1 | 1 |
| FOR each prerequisite | 1 | n | n |
| PRINT prerequisite | 1 | n | n |
| **Total Cost** | | | 3n + 1 |
| **Runtime** | | | O(n) |

**Hash Table Runtime Analysis**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| FOR each course in list | 1 | 1 | 1 |
| IF course == searchTerm | 1 | 1 | 1 |
| PRINT course data | 1 | 1 | 1 |
| FOR each prerequisite | 1 | 1 | 1 |
| PRINT prerequisite | 1 | 1 | 1 |
| **Total Cost** | | | 5 |
| **Runtime** | | | O(1) |

**Binary Search Tree Runtime Analysis**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| FOR each course in list | 1 | log n | log n |
| IF course == searchTerm | 1 | log n | log n |
| PRINT course data | 1 | 1 | 1 |
| FOR each prerequisite | 1 | n | n |
| PRINT prerequisite | 1 | n | n |
| **Total Cost** | | | 2 log n + 3 |
| **Runtime** | | | O(log n) |

**Vector**Vectors load data in O(n) time by storing courses in sequential order. While this makes inputting data fast, searching requires a linear approach which can become inefficient with larger data pools.

**Hash Table**Hash tables provide O(1) average search time using a hash function to jump directly to courses. However, because the data is stored without order, it will require an O(n log n) sort when printing the course list.

**Binary Search Tree**BSTs maintain automatic sorting through their O(log n) input and output structure. While slightly slower than hash tables for searching, their built-in sorting eliminates the need for separate O(n log n) tasks.

**Recommendation**The BST's O(log n) offers a great balance of input and output because it can run significantly faster than a linear vector approach for searches and utilize lesser memory allocation compared to hash tables since it doesn’t have to pre-allocate the stored data. With the support of the in-order traversal function, BSTs become more efficient overall.